EXPRESS [illegible]IL LABEL NO:
EL252575[illegible]S

# OPTIMAL INITIAL RASTERIZATION STARTING POINT

Lordson L. Yue
James T. Battle

## BACKGROUND OF THE INVENTION

An image typically includes several objects (e.g., a tree, sky or animated character). Each object may be computer represented by a group of triangles. Vertex data for each triangle includes x and y-coordinate data defining the position of each vertex of the triangle within the image. In three-dimensional applications, the vertex data also includes z-coordinate data which defines the depth of the triangle in virtual space. On a display, a triangle having a greater depth may be obscured by a triangle having a lesser depth, thereby giving the appearance that the image is three-dimensional.

In writing each image frame, a graphics processor feeds triangle data, one triangle at a time, to a rasterizer which assigns luminance and color values to each pixel location within the triangle. After all triangles of the image frame are written into a frame buffer, the image frame is displayed. In typical graphics applications, an image frame may include many thousands of triangles depending on the image resolution of the image frame. Furthermore, many image frames are displayed each second in graphics applications. Thus, as the graphics applications become more complex, the graphics processor and rasterizer must operate faster.

## SUMMARY OF THE INVENTION

Images may be represented as a group of triangles. A rasterizer assigns pixel values corresponding to one triangle at a time to a frame buffer, each triangle represented by vertex data. In one embodiment, a frame buffer is divided into tiles of, for example, 32 by 32 pixels. Triangles (and portions thereof) that are within a current tile are rasterized one triangle at a time into the tile location. This process repeats for each tile in the image frame. For graphics applications, this process repeats for each image frame in the graphics stream.

In accordance with the present invention, vertex data corresponding to three vertices of a triangle are received in a sorting circuit. The sorting circuit generates control bits representing an order of the vertices along, for example, the vertical direction. A multiplexer passes data corresponding to one (e.g., the highest) of the vertices in response to the control bits. Similarly, other multiplexers may pass data corresponding to the middle and lowest vertices. Thus, the vertices of the triangle are sorted through the multiplexers such that the ordering of the vertices after the multiplexer is predictable. This predictability simplifies the downstream logic circuit by reducing the vertex position permutations that the downstream logic circuit may encounter.

A region calculation circuit generates region bits representing a location of each of the vertices with respect to a current tile. A trivial discard of the triangle data occurs if the region bits indicate that the entire triangle lies outside of the tile. For example, a trivial discard occurs if the region bits

indicate that the lowest vertex is higher than the top edge of the tile, that the highest vertex is lower than the bottom edge of the tile, that all the vertices are left of the left edge of the tile, or that all the vertices are right of the right edge of the tile.

After the trivial discard, initial rasterization starting point estimate coordinates are generated for the rasterizer. This starting point estimate lowers the time needed for the rasterizer to find the first pixel of the current triangle to be assigned values. The starting point may be generated using the region bits.

The principles of the present invention will best be understood in light of the following detailed description along with the accompanying drawings.

## BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a schematic diagram of a setup engine according to the invention.

Fig. 2 is a detailed diagram of the initial rasterization starting point estimation circuit of Fig. 1.

Figs. 3A, 3B, 3C, 3D, 3E and 3F show various permutations of y-coordinate ordering of triangle vertices.

Fig. 4 shows the current tile and eight surrounding regions.

Fig. 5 shows four triangles that are trivially discarded and one which is not.

Fig. 6 shows a triangle that is trivially accepted.

Fig. 7 shows a triangle of which the highest vertex lies north of the current tile.

Fig. 8 shows a triangle of which line 0'1' is the highest line of the triangle to intersect LEFT_EDGE.

Fig. 9 shows a triangle of which line 1'2' is the highest line of the triangle to intersect LEFT_EDGE.

Fig. 10 shows a triangle of which line 0'2' is the highest line of the triangle to intersect LEFT_EDGE.

Fig. 11 shows a triangle of which line 0'1' is the highest line of the triangle to intersect RIGHT_EDGE.

Fig. 12 shows a triangle of which line 1'2' is the highest line of the triangle to intersect RIGHT_EDGE.

Fig. 13 shows a triangle of which line 0'2' is the highest line of the triangle to intersect RIGHT_EDGE.

DETAILED DESCRIPTION OF THE INVENTION

In this description, the same or similar elements in different drawings are identified with the same reference symbols. In this description, "&" means logical AND, "+" means logical OR, and "!" means logical NOT. Items within parentheses "()" have

highest logical priority followed by "!", "&", and "+", in descending priority order.

A frame buffer (not shown) may be divided into tiles of, for example, 32 by 32 pixels. Triangles (and portions thereof) that are within a given tile are rasterized one triangle at a time into the tile location. This process repeats for each tile in the image frame. For graphics applications, this process repeats for each image frame in the graphics stream.

Fig. 1 shows a setup engine 101 that receives triangle vertex data v0, v1 and v2 from a vertex data feeding circuit 100, and provides data including an initial rasterization starting point estimate ("IRSPE") to a rasterizer 102. Although the vertex data v0, v1 and v2 are shown transmitted over three separate buses in Fig. 1, the vertex data v0, v1 and v2 may also be transmitted serially over a single bus as is known in the art. Vertex data v0, v1 and v2 represent a corresponding vertex 0, 1 and 2 of a triangle 103 to be set up for rasterization. The vertex data v0, v1 and v2 include, but are not limited to, x-coordinate components v0.x, v1.x and v2.x and y-coordinate components v0.y, v1.y and v2.y, respectively. The vertex data v0, v1 and v2 may also include, z-coordinate, color, blending and texture data, and other data as desired.

Setup engine 101 transforms the vertex data v0, v1 and v2 by 1) sorting the vertex data v0, v1 and v2 in y-coordinate order, 2) trivially discarding vertex data of triangles entirely outside of the current tile, 3) estimating an initial rasterization starting point that minimizes the searching by rasterizer 102 required to find the rasterization starting point of the triangle

103, and 4) any other operations on v0, v1 and v2 as desired. Rasterizer 102 draws whatever portions of triangle 103 that are within a given tile into a frame buffer (not shown).

Setup engine 101 sorts vertex data v0, v1 and v2 in y-coordinate order to simplify the downstream logic circuit as the number of permutations the downstream logic circuit must deal with is reduced due to the predictability of the y-ordering. Of course, this ordering may be in descending or ascending y-coordinate order as long as the y-positioning of the vertex is made predictable. Furthermore, if the rasterizer rasterizes vertically, setup engine 101 may sort the vertices according to the x-coordinate. For clarity, in the example that follows, the vertices are described as being sorted in descending y-coordinate order.

A y-sort circuit 110 receives the y-coordinate data v0.y, v1.y and v2.y, and generates control bits y01, y02, and y12 which cause multiplexers 150, 151 and 152 to sort the vertex data v0, v1 and v2 in, for example, descending y-coordinate order. Bit y01 has a value 1 only if v0.y is greater than v1.y, bit y02 has a value 1 only if v0.y is greater than v2.y, and bit y12 has a value 1 only if v1.y is greater than v2.y. Table 1 summarizes the y-coordinate ordering given input bits y01, y02 and y12.

| case # | y01 | y02 | y12 | v0.y | v1.y | v2.y | Fig. |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | highest | middle | lowest | 3A |
| 1 | 1 | 1 | 0 | highest | lowest | middle | 3B |
| 2 | 1 | 0 | 1 | - | - | - | none |
| 3 | 1 | 0 | 0 | middle | lowest | highest | 3C |
| 4 | 0 | 1 | 1 | middle | highest | lowest | 3D |
| 5 | 0 | 1 | 0 | - | - | - | none |
| 6 | 0 | 0 | 1 | lowest | highest | middle | 3E |
| 7 | 0 | 0 | 0 | lowest | middle | highest | 3F |

Table (1)

The six possible cases 0, 1, 3, 4, 6 and 7 of Table (1) for the y-coordinate ordering of the three vertices are shown respectively in Figs. 3A, 3B, 3C, 3D, 3E and 3F.

In response to control bits y01, y02 and y12, multiplexer 150 passes the vertex data having the highest y-coordinate value (v0 in cases 0 and 1, v1 in cases 4 and 6, and v2 in cases 3 and 7) into memory 160. Multiplexer 151 passes the vertex data having the middle y-coordinate value (v0 in cases 3 and 4, v1 in cases 0 and 7, and v2 in cases 1 and 6) into memory 161. Multiplexer 152 passes the vertex data having the lowest y-coordinate value (v0 is cases 6 and 7, v1 in cases 1 and 3, and v2 in cases 0 and 4) into memory 162. Hereinafter, the vertex data in memories 160, 161 and 162 are respectively referred to as "highest vertex data V0", "middle vertex data V1" and "lowest vertex data V2" corresponding to the sorted vertices 0', 1' and 2' of triangle 103'. Y-sorting of the vertex data is thus completed.

After sorting, a trivial discard analysis occurs as follows. Fig. 4 shows the current tile TILE and eight surrounding regions 401-408. TILE is defined by edges TOP_EDGE, RIGHT_EDGE, BOTTOM_EDGE and LEFT_EDGE. Hereinafter, the x-coordinate position of LEFT_EDGE and RIGHT_EDGE is LEFT_EDGE.x and RIGHT_EDGE.x, respectively. The y-coordinate position of TOP_EDGE and BOTTOM_EDGE is TOP_EDGE.y and BOTTOM_EDGE.y, respectively.

Referring to Fig. 1, a region calculation circuit 120 reads the permuted x-coordinate data (V0.x, V1.x and V2.x), the sorted y-coordinate data (V0.y, V1.y and V2.y), and the tile boundaries TOP_EDGE.y, BOTTOM_EDGE.y, LEFT_EDGE.x and RIGHT_EDGE.x. Based on these values, region calculation circuit 120 generates region bits n0, n1, n2, e0, e1, e2, s0, s1, s2, w0, w1 and w2 representing the position of the sorted vertices 0', 1' and 2' with respect to TILE.

Referring to Fig. 4, vertex i' (where i is 0, 1 or 2) is considered north ($n_i = 1$) of TILE if its y-coordinate is greater than TOP_EDGE.y (regions 408, 401 and 402 of Fig. 4), east ($e_i = 1$) of TILE if its x-coordinate is greater than RIGHT_EDGE.x (regions 402, 403 and 404), south ($s_i = 1$) of TILE if its y-coordinate is less than BOTTOM_EDGE.y (regions 406, 405 and 404 of Fig. 4), and west ($w_i = 1$) of the tile if its x-coordinate is less than LEFT_EDGE.x (regions 408, 407 and 406 of Fig. 4).

An orientation circuit 130 generates a bit CW having a value 1 only if the line ("line 0'2'") connecting vertex 0' and 2' is oriented clockwise from the line ("line 0'1'") connecting vertex 0' and 1'. In

other words, orientation circuit 130 assigns a value 1 to bit CW only if Equation (1) is true.

$$(V0.x-V1.x)(V0.y-V2.y) < (V0.x-V2.x)(V0.y-V1.y) \quad (1)$$

Y-sort circuit 110, region calculation circuit 120 and orientation circuit 130 all have access to a comparator 140 to perform the above comparisons.

The region bits n0, n1, n2, e0, e1, e2, s0, s1, s2, w0, w1 and w2; the orientation bit CW; and the vertex data V0, V1 and V2 are inputted into IRSPE circuit 170 (Fig. 1) shown in further detail in Fig. 2. IRSPE circuit 170 may be, for example, a portion of a larger setup operational unit.

Referring to Fig. 2, IRSPE circuit 170 trivially discards the triangle 103 under certain conditions as shown in Table (2) in which the triangle lies completely outside of TILE.

| Case # | trivial discard if true | rationale for trivial discard |
|---|---|---|
| 1 | n2 = 1 | implies entire triangle is north of TILE (i.e., north of TOP_EDGE) |
| 2 | s0 = 1 | implies entire triangle is south of TILE (south of BOTTOM_EDGE) |
| 3 | w0&w1&w2 = 1 | implies entire triangle is west of TILE (i.e., west of LEFT_EDGE |
| 4 | e0&e1&e2 = 1 | implies entire triangle is east of TILE (i.e., east of RIGHT_EDGE) |

Table (2)

Cases 1, 2, 3 and 4 are shown respectively as triangles 501, 502, 503 and 504 of Fig. 5. If any one of cases 1

to 4 of Table (2) is true, trivial discard circuit 208 generates a bit y308 of a value 1, causing setup engine 101 (Fig. 1) to request the next triangle for TILE without providing data to rasterizer 102 (Fig. 1).

Note that bit y308 sometimes has a value 0 even though the triangle (e.g., triangle 505 of Fig. 5) lies entirely outside of TILE. Performing a discard step for all triangles that are entirely outside of TILE would be computationally intensive and might slow down the triangle setup procedure.

If none of the trivial discard cases are true (i.e., bit y308 has a value 0), setup engine 101 estimates an initial rasterization starting point. This reduces the number of cycles required for rasterizer 102 to find the first pixel that requires assignment of luminance and/or color values.

A typical rasterizer checks each pixel in a field (e.g., a tile or frame) to determine if the pixel is positioned within an object (e.g., a triangle). If not, the rasterizer proceeds to the next pixel and so on in a raster pattern until a pixel is found which is positioned in the object. Once a pixel within the object is found, there are numerous conventional ways to reduce the number of pixels checked by a rasterizer before the object is entirely rasterized. However, often significant time is taken finding a pixel that lies within the object. Setup engine 101 (Fig. 1) provides rasterizer 102 (Fig. 1) with initial rasterization starting point estimation coordinates IRSPE.x and IRSPE.y to reduce this time.

Specifically, referring to Fig. 2, IRSPE circuit 170 generates control bits y312, y316, y320, y324,

y328, y332, y336 and y340 causing multiplexers 260 and 270 to pass the estimate coordinates IRSPE.x and IRSPE.y as described below.

Trivial accept circuit 212 assigns a value 1 to bit y312 only if !n0&!e0&!s0&!w0 equals 1, in which case the highest vertex 0′ lies within the selected tile as in triangle 601 of Fig. 6. If bit y312 has value 1, multiplexers 260 and 270 pass the actual coordinates V0.x and V0.y of the top vertex 0 as IRSPE.x and IRSPE.y.

Circuit 216 assigns a value 1 to bit y316 only if n0&!e0&!w0 equals 1, in which case the highest vertex 0′ lies directly north of TILE as in triangle 701 of Fig. 7. Since there has been no trivial discard of triangle 103′, triangle 103′ does not lie entirely north of TOP_EDGE. Thus, the line 0′2′ must intersect with TOP_EDGE if bit y316 has a value 1. A bit y316 of value 1 causes an intercept calculation circuit 280 to output value INT representing the x-intercept of line 0′2′ with TOP_EDGE, the value INT being horizontally clamped right to LEFT_EDGE.x or left to RIGHT_EDGE.x if needed. A bit y316 of value 1 causes multiplexers 260 and 270 to pass INT as IRSPE.x and TOP_EDGE.y as IRSPE.y.

Circuit 220 assigns a value 1 to bit y320 only if w0&!w1&CW has a value 1, in which case the line 0′1′ is the highest line of the triangle to intersect LEFT_EDGE as in triangle 801 of Fig. 8. A bit 320 of value 1 causes the intercept calculation circuit 280 to assign, as value INT, the y-intercept of line 0′1′ with LEFT_EDGE, the value INT being vertically clamped down to TOP_EDGE if INT is higher than TOP_EDGE, or clamped up to BOTTOM_EDGE if INT is lower than BOTTOM_EDGE.

Hereinafter, this clamping is referred to as "vertical clamping". A bit y320 of value 1 causes multiplexers 260 and 270 to pass LEFT_EDGE.x as IRSPE.x and INT as IRSPE.y.

Circuit 224 assigns a value 1 to bit y324 only if w0&w1&CW equals 1. Since no trivial discard has occurred, the triangle does not lie entirely west of LEFT_EDGE. Thus, w2 must have a value 0. In this case, line 1'2' is the highest line of the triangle to intersect LEFT_EDGE as in triangle 901 of Fig. 9. A bit 324 of value 1 causes the intercept calculation circuit 280 to assign, as value INT, the vertically clamped y-intercept of line 1'2' with LEFT_EDGE. A bit y324 of value 1 causes multiplexers 260 and 270 to pass LEFT_EDGE.x as IRSPE.x and INT as IRSPE.y.

Circuit 228 assigns a value 1 to bit y328 only if w0&!CW equals 1. Since no trivial discard has occurred, line 0'2' is the highest line to intersect LEFT_EDGE as in triangle 1001 of Fig. 10. A bit 328 of value 1 causes the intercept calculation circuit 280 to assign, as value INT, the vertically clamped y-intercept of line 0'2' with LEFT_EDGE. A bit y328 of value 1 causes multiplexers 260 and 270 pass LEFT_EDGE.x as IRSPE.x and INT as IRSPE.y.

Circuit 232 assigns a value 1 to bit y332 only if e0&!e1&!CW equals 1, in which case the line 0'1' is the highest line to intersect RIGHT_EDGE as in triangle 1101 of Fig. 11. A bit 332 of value 1 causes the intercept calculation circuit 280 to assign, as value INT, the vertically clamped y-intercept of line 0'1' with RIGHT_EDGE. A bit y332 of value 1 causes multiplexers 260 and 270 pass RIGHT_EDGE.x as IRSPE.x and INT as IRSPE.y.

Circuit 336 assigns a value 1 to bit y336 only if e0&e1&!CW equals 1, in which case the line 1'2' is the highest line to intersect RIGHT_EDGE as in triangle 1201 of Fig. 12. A bit 336 of value 1 causes the intercept calculation circuit 280 to assign, as value INT, the vertically clamped y-intercept of line 1'2' with RIGHT_EDGE. A bit y336 of value 1 causes multiplexers 260 and 270 to pass RIGHT_EDGE.x as IRSPE.x and INT as IRSPE.y.

Circuit 240 assigns a value 1 to bit y340 only if e0&CW equals 1, in which case, since no trivial discard has occurred, the line 0'2' is the highest line to intersect RIGHT_EDGE as in triangle 1301 of Fig. 13. A bit 340 of value 1 causes the intercept calculation circuit 280 to assign, as value INT, the vertically clamped y-intercept of line 0'2' with RIGHT_EDGE. A bit y340 of value 1 causes multiplexers 260 and 270 pass RIGHT_EDGE.x as IRSPE.x and INT as IRSPE.y.

Thus, an initial rasterization starting point is estimated. Rasterizer 102 may now find the rasterization starting point faster using coordinates IRSPE.x and IRSPE.y. Note that assuming no trivial discard has occurred, exactly one of bits 312, 316, 320, 324, 328, 332, 336 and 340 has a value 1.

This process is repeated for all triangles within the selected tile, and for all tiles within the image frame. Although the above describes a specific embodiment of the present invention, this embodiment is illustrative only and not limiting. Various modifications and substitutions will be apparent to one skilled in the art. All such modifications and substitutions are intended to be part of the present

invention. The invention is defined by the following claims.